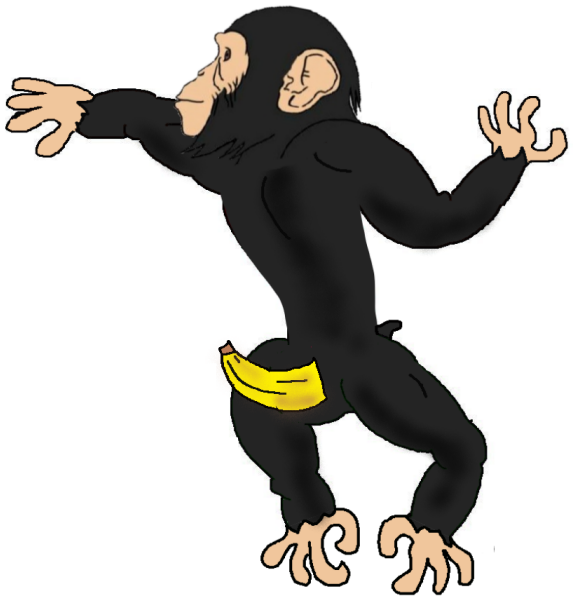
MONKES MANUAL
(DRAFT)

Francisco Javier Escoto López



1 Installation and required libraries

In this section it is explained how to install MONKES.

1.1 Fortran compiler

MONKES is written in Fortran language and thus requires having an installed Fortran compiler. This compiler must be compatible with LAPACK and NetCDF libraries. One possible compiler is the Intel Fortran compiler.

1.2 NetCDF library

In order to read BOOZER_XFORM output files, MONKES needs to use the NetCDF library. An example of the minimal required libraries for running MONKES in a UNIX machine is displayed below. Modules 1) and 2) are the NetCDF library compatible with the Fortran compiler. Module 3) is the Fortran compiler version, in this case intel17/17.0.8.

```
Currently Loaded Modules:
1) netcdf-fortran-4.4.4-intel-17.0.8-4me7upi
2) netcdf-4.6.1-intel-17.0.8-i5cn5xw
3) intel17/17.0.8
```

1.3 LAPACK library

For using LAPACK library there are two options. One can use the static or the dynamic version of the library. In listing 8, the dynamic case is illustrated.

```
# *** Compiler selection
f90comp=ifort
NETCDF_DIR=/mnt/lustre/ohpc/admin/spack/0.12.1/opt/spack/linux-centos7-x86_64/intel-17.0.8/netcdf-
    fortran-4.4.4-7aFYnuhdtreECAnwcH2MYPC7vfd7fUhEpv4r8ohz1zaL
FFLAGS   =-fpp -O3 -g -traceback -xHost -shared-intel -warn all -mcmodel large --with-debugging=yes
    --with-scalar-type=real --with-precision=double -r8 -start_in_debugger -heap-arrays -qopt-matmul
#
IFLAGS   =-I$(NETCDF_DIR)/include
LFLAGS   =-L$(NETCDF_DIR)/lib -lnetcdff -lnetcdf -L/${MKL_HOME}/lib/intel64 -mkl=parallel -Wl,-rpath
    ,${MKL_HOME}/lib/intel64
```

Listing 1: Makefile

The variable LFLAGS includes the linking of LAPACK libraries. Specifically, those flags which include the variable $MKL_HOME (variable containing the location of the dynamic library) set the instructions for where to find the library. The flag -mkl=parallel allows for LAPACK multithreading functionalities. In listing 3, it is shown how the monkes executable is generated.

```
main_monkes.x: $(objects)
  $(f90comp) -o main_monkes.x $(objects) ${COMP_LIB} ${LFLAGS} ${IFLAGS} ${FFLAGS}
```

Listing 2: Makefile

2 How to use MONKES

2.1 Magnetic configuration input

2.2 Monoenergetic database input

2.3 Running the executable with SLURM

2.4 Monoenergetic database output

3 Algorithm implementation

3.1 User level

```
! Select input type: BOOZER_XFORM output or DKES input
call Select_Input(input_case)
call Select_surface  ! Select flux surface (needed for BOOZER_XFORM output)

select case (input_case)

   case ("boozmn.nc")
   call READ_BOOZMNNC(s)

   case ("ddkes2.data")
   call read_ddkes2data("ddkes2.data")

   case default
   write(*,*) " There is no BOOZER_XFORM output 'boozmn.nc' &
               nor ddkes2.data in this folder. "

 end select
```

Listing 3: main_monkes.f90

```
call Monoenergetic_Database_Input
!call Monoenergetic_Database_Input_Orthonormal
```

Listing 4: main_monkes.f90

```
subroutine Monoenergetic_Database_Input
   integer, parameter :: N_nu_max = 500, N_E_r_max = 500
   integer, parameter :: M_theta_max = 500, M_zeta_max = 500, M_xi_max = 500
   real :: nu(N_nu_max), E_r(N_E_r_max)
   integer :: N_theta(M_theta_max), N_zeta(M_zeta_max) , N_xi(M_xi_max)
   integer :: N_nu, N_E_r, M_theta, M_zeta, M_xi, ierr
   namelist /parameters/N_theta, N_zeta, N_xi, nu, E_r

   ! *** Read input parameters from "monkes_input.parameters" file
    N_theta = -1 ; N_zeta = -1 ; N_xi = -1 ; nu = -1d14 ; E_r = -1d14
   open(1, file= "monkes_input.parameters", status="old")
   read(1, nml=parameters, iostat=ierr)
   close(1)

   ! Count number of collisionalities and radial electric field to be
   ! included in the database
   M_theta = count(N_theta > 0)
   M_zeta = count(N_zeta > 0)
   M_xi = count(N_xi > 1)
   N_nu = count(nu > 0) ; N_E_r = count(E_r /= -1d14)

   ! Adjust theta and zeta resolutions to be nearest odd number
   where( mod(N_theta(1:M_theta),2) == 0 )  N_theta(1:M_theta) = N_theta(1:M_theta) + 1
   where( mod(N_zeta(1:M_zeta),2) == 0 )    N_zeta(1:M_zeta) = N_zeta(1:M_zeta) + 1

   call Monoenergetic_Database_Scan( nu(1:N_nu), E_r(1:N_E_r),       &
```

```
                                        N_theta(1:M_theta),             &
                                        N_zeta(1:M_zeta),               &
                                        N_xi(1:M_xi) )

  end subroutine
```

Listing 5: `examples/API_Example_DKE_BTD_Solution_Legendre.f90`

The subroutine `Monoenergetic_Database_Scan` computes the monoenergetic database by looping in the different parameters. For this, it calls within the loop the subroutine `Solve_BTD_DKE_Legendre`. What the routine `Solve_BTD_DKE_Legendre` does is out of the scope of this section (see Developer section). The output is written in the file `monkes_Monoenergetic_Database.dat`

```
  subroutine Monoenergetic_Database_Scan( nu, E_r, N_theta, N_zeta, N_xi, DD, DD_33_Sp  )
    real, intent(in) :: nu(:), E_r(:)
    integer, intent(in) :: N_theta(:), N_zeta(:), N_xi(:)
    real, optional, intent(out) :: DD( 3, 3, size(nu), size(E_r), size(N_theta), size(N_zeta), size
   (N_xi) )
    real, optional, intent(out) :: DD_33_Sp( size(nu), size(E_r), size(N_theta), size(N_zeta), size
   (N_xi) )

    real :: D( 3, 3, size(nu), size(E_r), size(N_theta), size(N_zeta), size(N_xi) )
    real :: D_33_Sp( size(nu), size(E_r), size(N_theta), size(N_zeta), size(N_xi) )

    integer :: i, j, ii, jj, kk, N_nu, N_E_r, M_theta, M_zeta, M_xi
    character(len=500) :: file_path
    real :: t_clock, rate, t0, t1
    integer :: c0, c1, c_rate

    ! Initialize compiler internal clock for computing wall-clock time
    call system_clock(count_rate=c_rate) ; rate = real(c_rate)

    ! Vectors sizes
    N_nu = size(nu) ; N_E_r = size(E_r)
    M_theta = size(N_theta) ; M_zeta = size(N_zeta) ; M_xi = size(N_xi)

    ! Location for output
    file_path = "monkes_Monoenergetic_Database.dat"
    ! Open output file and write header
    open(21, file=trim(file_path))
    write(21,'(9999A25)') " nu/v [m^-1]", " E_r/v [kV s /m^2]", &
                          " N_theta ", " N_zeta ", " N_xi ", &
                          " D_11 ", " D_31 ", &
                          " D_13 ", " D_33 ", &
                          " D_33_Spitzer ",   &
                          " Wall-clock time [s] ",   &
                          " CPU time [s] "
    do j = 1, N_E_r ! Loop electric field value
       do i = 1, N_nu ! Loop collisionality value
          do kk = 1, M_xi ! Loop number of Legendre modes
             do ii = 1, M_theta ! Loop number of theta points
                do jj = 1, M_zeta ! Loop number of zeta points

                   call system_clock(c0) ; call cpu_time(t0)
                   call Solve_BTD_DKE_Legendre( N_theta(ii),          &
                                                N_zeta(jj),           &
                                                N_xi(kk),             &
                                                nu(i), E_r(j),        &
                                                D(:,:,i,j,ii,jj,kk),  &
                                                D_33_Sp(i,j,ii,jj,kk) )
                   call system_clock(c1) ; call cpu_time(t1)

                   ! Wall-clock time in seconds
                   t_clock = ( c1 - c0 ) / rate

                   ! Writing results on "monkes_Monoenergetic_Database.plt"
                   write(21,'(9999e)') nu(i), E_r(j), &
```

```
                                        real(N_theta(ii)), &
                                        real(N_zeta(jj)), &
                                        real(N_xi(kk)), &
                                        D(1,1,i,j,ii,jj,kk), &
                                        D(3,1,i,j,ii,jj,kk), &
                                        D(1,3,i,j,ii,jj,kk), &
                                        D(3,3,i,j,ii,jj,kk), &
                                        D_33_Sp(i,j,ii,jj,kk), &
                                        t_clock, t1-t0
                end do
              end do
            end do
          end do
        end do
        close(21)

        ! Monoenergetic coefficients as output
        if ( present(DD) ) DD = D
        if ( present(DD_33_Sp) ) DD_33_Sp = D_33_Sp
    end subroutine
```

Listing 6: examples/API_Example_DKE_BTD_Solution_Legendre.f90

3.2 Developer level: main routines

3.3 Developer level: libraries

4 Application Programming Interface (API)